MINISTRE DE L'ENSIEGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABEDRAHMAN MIRA Targa Ouzemour Bejaia

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Faculté Science Exactes

Département Informatique

Rapport de travail

Thème

Prédiction de la résiliation des contrats d'une entreprise de télécommunication

Réalisé par :

M. MENASERIA Mohamed

Mlle. SADOUN Kenza

Mlle. ALLIK Dalila

Encadré par :

Mme.

17 Mai 2023

Table de matières

Introduction générale

De nos jours, le Machine Learning est un domaine qui permet aux ordinateurs d'apprendre et de s'améliorer à partir de données sans être explicitement programmés. Dans le domaine des télécommunications, son application offre de nombreux avantages pour la prédiction et l'analyse des données. Ses algorithmes tels que la régression et les arbres de décision, etc… permettent de modéliser et d'analyser les données en temps réel, offrant ainsi aux entreprises une vision prédictive importante pour anticiper les tendances du marché, optimiser les réseaux, améliorer la qualité du service et mieux comprendre les besoins des clients.

1. Travail et objectifs

Notre travail consiste à construire deux modèles autrement dis algorithmes de machine learning pour la prédiction de la résiliation des contrats des clients, les entrainer, les évaluer et prédire sur le jeu de données est sur la plateforme kaggle. Le but de ce travail est fournir à l'entreprise des prédictions précises dans le but est de maximiser son bénéfice et apporter des recommandations pour l'aide à la décision afin améliorer sa stratégie.

2. Régression logistique

La régression logistique est une technique d'apprentissage automatique utilisée pour modéliser et prédire des variables binaires ou catégoriques. Elle est couramment utilisée pour estimer la probabilité d'appartenance à une classe en fonction de variables explicatives. Elle utilise une fonction logistique pour transformer une combinaison linéaire des caractéristiques en une probabilité, permettant ainsi une classification précise dans des problèmes de classification binaire.

2.1 Modèle construit

- Premièrement nous allons importer les bibliothèques dont nous avons besoin et le Dataset aussi :

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
```

```
[2]: dataset = pd.read_csv("ds_prt.csv", delimiter=';')
     print(dataset)
```

- Ensuit Nous faisons la sélection de caractéristiques et les étiquetées :

```
x = dataset.iloc[0:, :14]
x
```

```
y= dataset.iloc[0:,14]
y
```

- Après nous découpons le Dataset à une partie de traitement et la partie de teste et nous augmentons le X par une colonne de 1.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=45)
print(y_train)
y_train[10]
```

```
[1. 1. 0. ... 0. 0. 0.]
0.0
```

```
X_train = np.c_[np.ones(X_train.shape[0]), X_train]
```

- Nous donnons des valeurs aléatoires à Thêta et définissons les fonctions Sigmoïde et fonction cout et descend de gradient

```
theta = np.random.randn(X_train.shape[1])
print(theta)
```

```
[-0.69699705 -2.42855057 -0.6026026   1.48862103  1.30341504  0.09132358
 -0.08194904 -0.57188069 -1.03321877  2.41381638 -0.07045923 -1.57852785
  0.07561015 -2.16026351  0.12675107]
```

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

```
def cost_function(X, y, theta):
    h = sigmoid(X.dot(theta))
    m = len(y)
    J = -1/m * (y.T.dot(np.log(h)) + (1-y).T.dot(np.log(1-h)))
    return J
```

```
def gradient_descent(X_train, y_train, theta, alpha, num_iters):
    m = len(y_train)
    J_history = np.zeros(num_iters)

    for i in range(num_iters):
        h = sigmoid(X_train.dot(theta))
        theta = theta - alpha/m * X_train.T.dot(h - y_train)
        J_history[i] = cost_function(X_train, y_train, theta)

    return theta, J_history
```

- Ensuite nous passons à faire le test sur la partie de test pour faire de prédiction et on calcule la précision et l'erreur

```
R=0
A=0
for i in range (len(y_pred)):
   if (y_pred[i]==0)and(y_test[i]==0):
    R=R+1
for i in range (len(y_pred)):
   if (y_pred[i]==1)and(y_test[i]==1):
    A=A+1
precision=A+R
```

```
precision
```

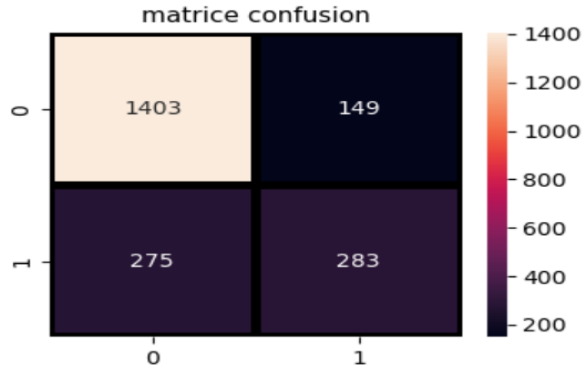
```
1686
```

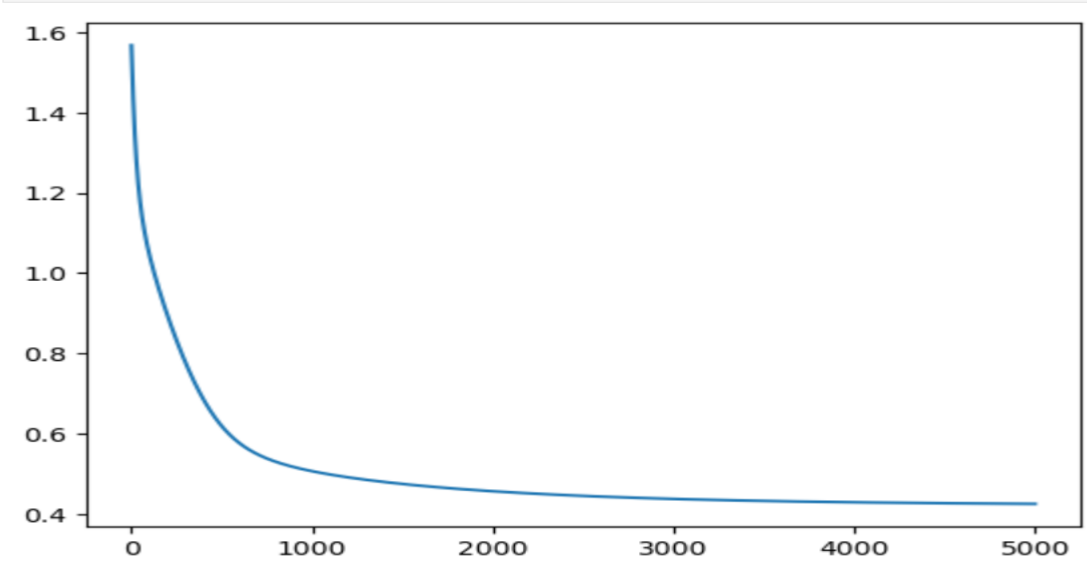
```
taux_precision= precision/len(y_test)
taux_precision
```

```
0.7990521327014218
```

- Matrice de confusion

```
#matrice de confusion
import seaborn as sn
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(4,3))
sn.heatmap(confusion_matrix(y_test,y_pred), annot=True,fmt = "d",linecolor="k",linewidths=3)

#affichage
plt.title("matrice confusion",fontsize=11)
plt.show()
```



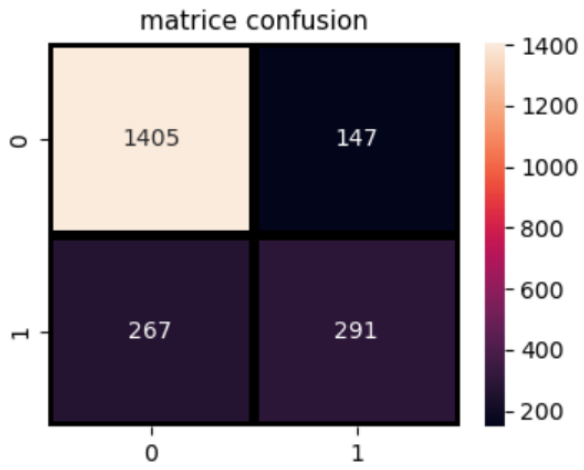


- SKLEARN :

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score



# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred2 = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred2)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8037914691943128
```

```
plt.figure(figsize=(4,3))
sn.heatmap(confusion_matrix(y_test,y_pred2), annot=True,fmt = "d",linecolor="k",linewidths=3)

#affichage
plt.title("matrice confusion",fontsize=11)
plt.show()
```



2.3 Evaluation

```
R=0
A=0
for i in range (len(y_pred)):
    if (y_pred[i]==0)and(y_test[i]==0):
     R=R+1
for i in range (len(y_pred)):
    if (y_pred[i]==1)and(y_test[i]==1):
     A=A+1
precision=A+R
```

```
precision
```

```
1686
```

```
taux_precision= precision/len(y_test)
taux_precision
```

```
0.7990521327014218
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# Create a logistic regression model
model = LogisticRegression()
# Train the model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred2 = model.predict(X_test)
# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred2)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8037914691943128
```

- On remarque qu'en utilisant les fonctions prédéfinit (sklearn), avec la métrique « matrice de confusion », le modèle construit est très comparable avec celui déjà définit en terme de performance. Donc notre modèle peut nous fournir une bonne prédiction.

3. Classification Bayésienne naïve

La classification bayésienne naïve est un algorithme supervisé basé sur le théorème de Bayes. Il suppose une indépendance conditionnelle entre les caractéristiques, il calcule la probabilité qu'un objet appartienne à une classe spécifique en utilisant la probabilité a priori de cette classe et les probabilités conditionnelles des caractéristiques données. Dans notre cas on a choisi de travailler avec la loi normale.

3.1 Modèle construit

```python
#probabilite cond (gauss)
def pro_cond(X, moyenne, ecart_type):
    exponent = np.exp(-((x - moyenne) ** 2) / (2 * (ecart_type ** 2)))
    return (1 / (np.sqrt(2 * np.pi) * ecart_type)) * exponent
```

```python
#classif bayes
def modele(xtrain, ytrain):
    nb_classes = 2
    caract = xtrain.shape[1]
    moyennes = np.zeros((nb_classes, caract))
    ecart_types = np.zeros((nb_classes, caract))
    prioris = np.zeros(nb_classes)

#moy et ecart type de chaque classe
    for classe in range(nb_classes):
        xclasse = xtrain[ytrain == classe]
        moyennes[classe] = xclasse.mean(axis=0)
        ecart_types[classe] = xclasse.std(axis=0)

#nouvelle prob
    for classe in range(nb_classes):
        prioris[classe] = np.sum(ytrain == classe) / len(ytrain)

    return moyennes, ecart_types, prioris
```

- La première fonction « pro_cond », calcule la probabilité conditionnelle avec la loi normale, qui dépend de l'écart-type de la variable ainsi que sa moyenne.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \, \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

- La deuxième représente notre modèle de classification bayésienne, « moyennes » et « ecart-types » sont des matrices pour stocker les variables, « prioris » est un tableau pour stocker les probabilités initiales.
- La première boucle for, calcule la moyenne et l'écart-type de chaque classe {0,1}, pour la deuxième elle calcule les nouvelles probabilités.

3.2 Prédictions

```
def predire(xtest, moyennes, ecart_types, prioris):
    nb_classes = len(prioris)
    nb_test = xtest.shape[0]
    predictions = np.zeros(nb_test, dtype=int)

    for i in range(nb_test):
        probabilites_classes = np.zeros(nb_classes)

        for classe in range(nb_classes):
            probabilites_caract = pro_cond(xtest[i], moyennes[classe], ecart_types[classe])
            probabilites_caract = np.array(probabilites_caract, dtype=float)
            probabilites_classes[classe] = np.prod(probabilites_caract) * prioris[classe]

        predictions[i] = np.argmax(probabilites_classes)

    return predictions
```

- La fonction prédire permet de réaliser une prédiction sur l'ensemble de test, calcule la probabilité premièrement pour chaque échantillon puis pour chaque classe et enfin il fait la sélection de la probabilité max.

3.3 Evaluations

```
pred = predire (xtest, moyennes, ecart_types, prioris)
```

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(ytest, pred)
print("Précision :", accuracy)
```

```
Précision : 0.7313432835820896
```

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(xtrain, ytrain)
y_pred = model.predict(xtest)
print(y_pred)

#precision
print("Précision:", accuracy_score(ytest, y_pred))
```

```
[1. 0. 1. ... 1. 1. 0.]
Précision: 0.7107320540156361
```

- On remarque qu'en utilisant les fonctions prédéfinit (sklearn), avec la métrique « accuracy-score » c'est-à-dire la précision, le modèle construit est très comparable avec celui déjà définit en terme de performance. Donc notre modèle peut nous fournir une bonne prédiction.

4. Interprétations

Pour la régression logistique on observe une précision de 0.79 et pour la classification bayésienne naïve on a une précision de 0.73. On a d'un côté la régression logistique utilise la descente de gradient, en revanche la classification bayésienne utilise la loi normale. De ce fait on peut déduire que le modèle le plus adéquat pour le dataset dont on dispose est la régression logistique.

Conclusion générale

A la fin de notre travail, on a conclu que les algorithmes de machine Learning sont très effaces pour la prédiction, et sont de grande aide pour plusieurs entreprises, ils leurs offrent une nouvelles vision sur leurs stratégies afin de maximiser leur bénéfices.